View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Munich RePEc Personal Archive

Munich Personal RePEc Archive

An Algorithm for the Simulation of Bounded Rational Agents

Stephan Schuster

University of Surrey

27. June 2009

Online at http://mpra.ub.uni-muenchen.de/15942/
MPRA Paper No. 15942, posted 30. June 2009 00:17 UTC

An Algorithm for the Simulation of Bounded Rational Agents

Stephan Schuster

University of Surrey, Department of Economics,
Guildford GU2 7XH, United Kingdom
Stephan.Schuster@surrey.ac.uk

Abstract. Non-classical models of economic behaviour, usually summarised under the notion of 'Bounded Rationality' criticise the assumptions of the standard economic model - hyperrationality, perfect and costless information, and unlimited mental processing capabilities. However, alternative approaches have either remained very simple or purely descriptive. Here, a computational approach is presented based on Simon's [13], [14] concept of bounded rationality and satisficing as a compromise between the oversimplification of analytical and the descriptiveness of rich cognitive models.

Key words: agent based modelling, bounded rationality, reinforcement learning, rule extraction

1 Background

The perfectly informed and rational homo oeconomicus has often been criticised as too unrealistic - humans would not have the computational power to calculate the best decisions, taking into account all information and all possible outcomes. Already Simon [14] argued to use simpler, psychologically more plausible algorithms. While the argument of bounded rationality is often use as critique of the standard economic model, the concept remains, however, vague [15]. Psychologists (e.g., [4], [8]) point out that most alternative models are still based on the fundamental assumption that expected utility and Bayesian reasoning are the basis for all human decision making under uncertainty. For example, subjective expected utility theory acknowledged that individuals are not fully informed and replaced objective probabilities with subjective, however, the basis for reasoning remained the same. Similarly, Prospect Theory [7] modified perfect rationality by stating that human decisions are biased by the anticipation of future losses instead of gains, and explain by this deviations from the maximising principle.

More recently, behavioural approaches have been applied in the social sciences. For example in game theory, Erev and Roth tested the predictive capabilities of simple reinforcement learning models by comparing theoretical results with real-world experiments [3], [10]. Camerer and Ho's 'Experience Weighted

Attraction' learning model (EWA) combines reinforcement learning and limited foresight in games [2]. Classifier systems approaches have been used in Agent-based computational Economics (e.g., [6]). Also, the cognitive architecture CLARION has been successfully applied to social simulation [17].

This paper contributes to the behavioural approaches. Simon's bounded rationality concept is used as theoretical framework. It is structured as follows: First, Simon's framework is shortly described, then the main principles of the bounded rationality algorithm (BRA) are presented, before being specified formally. A simple example demonstrates how the algorithm works in practice. Finally, BRA is compared with the most closely related approaches, which have been identified as the cognitive architecture CLARION and classifier systems.

2 Theory

A common problem that only few authors have attempted to resolve is the integration of cognition on the one end of the spectrum, and learning by experience on the other end. In this paper it is argued that a helpful starting point and a rich enough concept for such integration is already present in Simon's concepts [13], [14]. His framework is based on the following components:

- The set of behaviour alternatives A
- The set choice alternatives A' for bounded rational or computationally less powerful individuals; this set may be only a subset of A.
- Possible future states S
- Payoffs connected with S, represented as a function of S, $V(s)$.
- Probabilities for S. There is uncertainty which sate occurs after a particular behaviour, i.e. there may be more than one.

Bounded rational individuals do not typically know the mapping from behaviour alternatives A to future welfare $V(s)$. A possible strategy to learn about the occurrence and the desirability of these future states is according to Simon: Start with a mapping of each action alternative $a \in A$ to the whole set of S. Using a utility function such as $V(s) \in \{-1, 0, +1\}$, find $S' \subset S$ such that (expectedly) $V(s) = 1$. Then gather information to refine the mapping $A \rightarrow S'$ (i.e., which actions lead to which result under certain conditions) and search for feasible actions $A' \in A$ that map to S' [14]. In other words, an agent's goal is to find the states which satisfies its needs, by exploring the state-action space by applying alternative behaviours.

3 Principles of a Bounded Rationality Algorithm

The translation of Simon's framework into an executable algorithm can be captured best with the concept of mental models. A mental model is an internal representation of an external reality. The agent builds it using experience, its perception, and its problem-solving strategies. A mental model contains minimal information, is unstable and subject to change and used to make decisions

in novel circumstances. A mental model must be 'runnable' and able to provide feedback on the results. Humans must be able to evaluate the results of actions or the consequences of a change of state [9]. An agent is only interested in its own welfare and its goal is to find suitable behaviour strategies that optimise utility under different conditions. Information processing and memory is costly, so that the internal model being built has to be minimal and efficient with respect to the agent's own welfare. The main principles an algorithm has to account for can roughly be summarised as follows:

Evaluating cognitive cues In any state of the environment, the agent must be able to choose an action. If low or even negative rewards are experienced, the agent can attempt to apply a different action. If this fails to improve the agent's welfare, this is a hint to pay attention to more cues from the environment and distinguish better between different situations.

Deciding what to know Paying attention to all cues from the environment is computationally too expensive and memory too limited; humans must filter out certain aspects of their perception in order to decide and act effectively. The agent has to 'decide what to know' [11]. Since the measurement of useful information is the agent's welfare generated by its actions, this decision procedure is a search over all possible state-action mappings. If the agent is satisfied with a mental model containing a subset of these mappings, it might stop searching for a better model or decrease its search intensity.

Updating a cognitive model If the environment changes, some aspects of the internal model might become obsolete. The agent will then experience a change in utility. In certain states, learning a new behaviour might be sufficient. However, it might also be that the representation of the state is not accurate anymore (e.g. a new type of agent appears). In this case the representation has to be changed, e.g. by removing old representations and start the search process anew for certain parts of the model.

4 The Algorithm

The basic idea of BRA is to build an internal, flexible model of the environment the agent lives in. The environment is accessible by the input state s defining the current 'situation' the agent is in. The input state is matched with an internal symbolic representation $C_i \in C = \{C_1 \dots C_n\}$ of the state. The agent then chooses an action according to the general form $r_i : C_i \rightarrow A$. A is the action set, C is the set of all possible conditions that can be generated from the input dimensions, and C_i is a collection of conditions derived from C.

The next paragraphs develop the algorithm in some detail. A compact description in pseudocode is given in the appendix.

Reinforcement Learning Reinforcement Learning (RL, e.g. [18]) is used to implement the dynamic aspect of knowledge generation in the model. In each state agents learn by trial and error which action to apply in a given state. Successful actions are rewarded. Actions which yield a higher reward are selected with a high probability in the future, whereas bad actions, receiving a lower reward, are selected less often. The history of these reinforcements is summarised as action strength q. Whenever an action a has been applied, the strength is updated with the reward $p(t)$ observed for that action by the following equation:

$$q(a_t) = q(a_{t-1}) + \gamma(p(t) - q(a_{t-1})) \tag{1}$$

Depending on the parameter γ, the action-value function updates the strength of the current action based on the weight γ of previous experiences and the current reward. For a value $\gamma = 0.5$, for example, and reasonably large t this function approximates the average payoff generated with action a. The smaller γ, the stronger the impact of past experiences; conversely, for $\gamma = 1$ only the reward of the last action is considered, and all previous experiences discarded.

In the next step, the action probability is calculated according to the selection function:

$$pr(a_{i,t+1}) = \frac{e^{q(a_i)*\alpha}}{\sum_{j,j\neq i} e^{q(a_j)*\alpha}} \tag{2}$$

This selection function, also known softmax policy in RL, determines each action's selection probability depending on its own strength relative to the strengths of the alternative actions. The parameter α, $0 < \alpha < 1$, is a learning parameter that determines the rate of exploration. The larger α the less the influence of the action strength on the selection probability.

State space partitioning Learning by doing as described above happens for a given state s. This section describes how states are represented and perceived in the agent's internal world model.

Representation and search paths The state s is represented internally as a collection of attributes $\{att_1 \dots att_i\}$. Each attribute can have a number of possible values, for example nominal values such as 'low' or 'high', or numerical ranges, e.g. 0-1000. Attributes are connected by simple predicate logic. For example the predicate '(profit=low *or* profit=medium *or* profit=high) *and* (sales $0 <$ sales < 1000)' could describe the situation of a firm in the dimensions profit and sales. This representation is called a 'state descriptor', and formally denoted C_i. To each state descriptor actions are bound from which the action policy for this state can be learnt. In the firm example, actions could be an array of price levels. This binding constitutes formally the mapping $r_i : C_i \rightarrow A$.

The agent starts with a model covering all possible states. This initial state entails all attributes with their value spaces, thus the coarsest representation possible. In consecutive time steps, specialisations are developed stepwise by the application of a heuristic search method. For this, the space of state descriptions

is represented as a tree, where nodes at higher levels contain coarser, and nodes at deeper level of the tree finer mappings. Finer grained descriptions are 'expanded' from the predicates at higher levels. Which descriptions are expanded depends on a heuristic evaluation function, which here is the agent's utility. The task of the search process is thus to find the level of detail that describes the environment in such a way that generates the highest welfare for the agent.

State expansion mechanism Before the internal model is updated, the agent acts in its environment over a period μ. During this period, the value of existing state descriptions $R = \{r_1 \dots r_n\}$ is updated using feedback from the environment. After each μ steps, the state expansion mechanism is applied: First the node r_{expand} with the highest value on the search path is selected. If the search path is empty, the root node is selected. From there, the next level of the tree is expanded by partitioning the value spaces of the attributes constituting the conditions of r_{expand}. For attributes having discrete values, one value is picked randomly. Attribute values representing numeric ranges are split in half. For each partitioned attribute a new condition is created containing the partitioned attribute values or value range, and the remaining original attribute values (i.e. the number of successor nodes equals the number of attributes $\times$ 2 in the original condition). The conjunction of the predicates of the resulting level (after reduction) is equivalent to the expression of the parent node. By mapping A to each newly created condition set the new descriptors R' are generated. The path from each $r' \in R'$ up to the root node is added to the search path (without duplicates). The conjunction of state descriptions with no children in the search path is then equivalent to the initial state description. The RL mechanism selects actions only from the matching descriptor in the search path, so that there is always exactly one state description activated and one action selected at a time.

For example, going back to the firm example above, of the initial, exhaustive description $C'_{initial}$ = (profit=low *or* profit=medium *or* profit=high) *and* (0 < sales < 1000) the attribute profit is selected, and of its value range the value 'high'. The value space of the attribute is divided into the expression 'profit=low or profit=medium' and 'profit=high', respectively. The resulting specialised state descriptions are C'_1 = (profit=low *or* profit=medium) *and* (0 < sales < 1000) and C'_2= (profit=high) *and* (0 <sales < 1000). Analogously, the sales attribute is split in two intervals and two successor descriptors generated, so that four successor descriptors are created.

Model specialisation and generalisation With the state expansion mechanism it is possible to specialise the conditions in the state-action space in many ways. A heuristic evaluation function determines the direction of this process. This function is calculated as follows: First, the value of a state at time t is calculated as

$$v(r,t) = v(r,t-1) + \frac{1}{2}(q(a_t) - v(r,t-1)) \tag{3}$$

where $q(a_t)$ is the reward of the executed action in the state described by r. The function approximates an average of the state description value.

Before an expansion happens, some constraints have to be satisfied: A parameter χ limits the maximum number of nodes the tree can have, i.e. the maximum number of situations the agent can differentiate. New states can only be evolved at the cost of 'forgetting' other state descriptions (see below for deletion). Furthermore, since the deletion of nodes might occur, it is possible that state descriptions that were deleted are expanded again, so that endless cycles of generalisation and specialisation occur. The right balance has to be found depending on the stability of the environment; preventing many visits of identical descriptions too early can be harmful if the environment changes; on the other hand it binds valuable resources in the agent's mental processing. To tune this balance, a function with a cost parameter $\zeta, 0 < \zeta \leq 1$ is used to compute a value determining whether the successor description should be developed or not: The better a state descriptor compared with the average performance (measured by the average reward g) and the smaller ζ, the more frequent (recurrent) expansions beginning from that state descriptor are allowed (equation 4).

$$expand(r) = \begin{cases} true, & \text{if } expansions(r) = 0 \text{ or} \\ & \zeta \times expansions(r) \times g < v(r,t) \\ false, & \text{otherwise} \end{cases} \tag{4}$$

A state description might lead to a good solution strategy, but if only rarely visited is only of limited value (they only use up scarce memory space and processing capabilities). Therefore, a heuristic function h used by the process is the state-value weighted by the number of its activations to account for the recency of the value:

$$h(r,t) = v(r,t)\frac{activations(r)}{t} \tag{5}$$

The search process selects the node with the maximal heuristic $h(r,t)$ in the search path, if the *expand* condition is satisfied.

Before new states are developed after μ steps, the state descriptions of the current level may be deleted if they did not outperform the value of their parent states (performance could be, e.g., the average of the state description values). Analogously to rule specialisation, this generalisation process sets in after a certain time $\nu, \nu < \mu$.

Avoiding local search optima The mechanism may end up after a number of expansions at a level of the tree with a particular configuration of descriptors in the search path. There is no back-propagation of values, e.g. an update of the successor states with a discounted value of the current state, so that more general descriptions higher up in the tree or in other partially developed paths can have higher, although outdated values. If such higher historical values exist, this fact is used as a hypothesis that the current search path has become suboptimal due to a changed environment and that different paths should be explored. The process may therefore switch with probability $\rho, 0 \leq \rho \leq 1$ to a different, higher valued node in the tree and continue the expansion from there. The path to this node becomes thereby the search path.

5 An example

To demonstrate the principle, a simple bargaining game was simulated using the algorithm. The idea of bargaining games is that 2 players have to agree on a share after a finite number of offerings. If haggling takes too long, both players get nothing. Here, a simplified version of such games with discrete shares simulated. In this game, agents can demand a low, medium or high share of a good. Table 1 shows the payoffs. This distribution of payoffs leads to situation where demanding a low share guarantees a certain, but low payoff, while demanding a high share may yield a higher, but uncertain payoff.

In the first experiment, there are $N + 1$ agents: $N/2$ agents always demand the highest share, $N/2$ always the lowest. One agent has no predefined strategy, but learns what share to demand from encounters with other players. Agents demanding a low share are green, agents demanding a high share are blue. Each time step, agents are paired randomly and play their strategy. With each encounter the learner is told which colour the opponent has. The agent can then use this information to build a state-action tree. In the second experiment, strategies are assigned randomly to the green and blue agents. Simulations were run with $N = 10$ (i.e. the learner encountered with equal probability a green or blue agent) for 1000 steps. The model parameters were set at $\gamma = 1$, $\zeta = 0.4$, $\rho = 0.3$, $\mu = 25$, $\nu = 19$. The aim of this small experiment is only to demonstrate the

	low	medium	high
low	0.3,0.3	0.3,0.5	0.3,1
medium	0.5,0.3	0.5,0.5	0,1
high	1,0.3	0,0	0,0

Table 1. Payoffs of the Demand Game. The first number in a cell is the payoff of the row player, the second number the payoff of the column player.

working of the algorithm, not to explain bargaining behaviour. Therefore, only the evolution of the state tree of the learner is analysed.

Figure 1 shows the result of the tree-building process: The agent has learnt that it is beneficial to distinguish between the colours of opponents. When it meets green agents it demands a high share of the good, while it demands a low share if blue agents are encountered. The process thus converges to the optimal solution; in about 90 % of encounters with each type of agents the maximum possible payoff is obtained. With only two possible states, this distinction is easy to learn, an thus discovered early in the simulation (in only about 50 steps out of 1000 the initial state description 'opponent is blue or opponent is green' is used).

Figure 2 shows the result if the colours are assigned randomly: Since there is nothing to gain from a distinction of colours, the agent does not pay attention

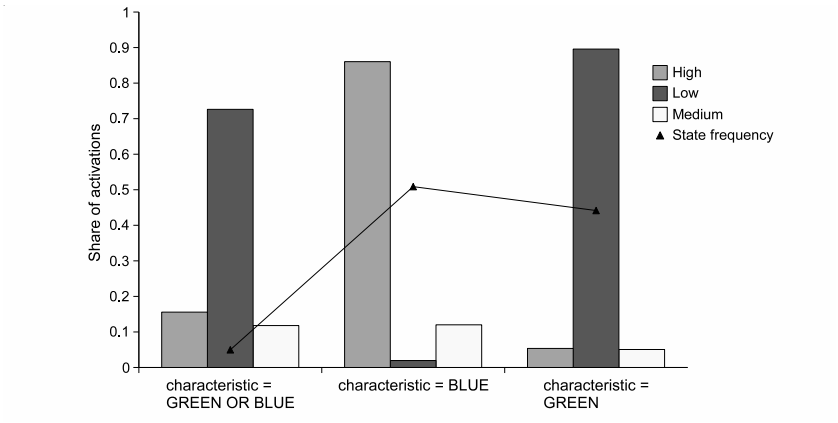

Fig. 1. Experiment 1. Colours correspond to actual strategies of the agents. The values are fractions of total activations of actions and encounters of state descriptions, respectively.

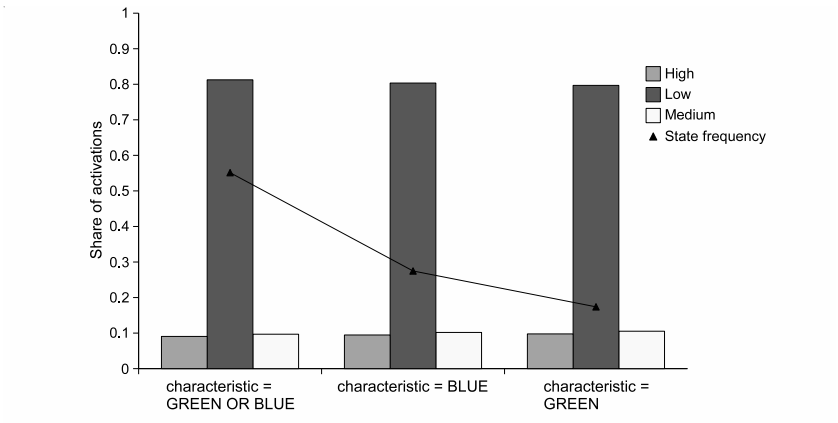

Fig. 2. Experiment 2. Colours are assigned randomly to strategy types. The values are fractions of total activations of actions, and encounters of state descriptions, respectively.

to this attribute. As a result, the agent always demands the low share irrespective of the other player's colour. Also, the most frequent state description is the initial state with no differentiation between colours. Here, the process converges to a result worse than the best response of a fully informed, perfect rational player with unlimited memory (as for strategy 'low' $\lim_{t\to\infty} p(t) = 0.3$; for strategy 'medium' $\lim_{t\to\infty} p(t) = 0.25$; and for strategy 'high' $\lim_{t\to\infty} p(t) = 0.5$). The reason for this is that γ is set to 1, so the agent considers only the last action result. γ was set to 1 because it supports an efficient action selection in the first experiment. The agent receives an error in strategy immediately (there is only one best response per colour), so smaller γ would add more noise and lead to slower convergence. In the second experiment this however leads to avoiding any strategy with generating low payoffs in the short run. This also illustrates that there is no perfect choice of parameters for any type of environment.

6 Related approaches

There are many ad-hoc, model-specific learning mechanisms in the agent-based modelling literature which are not discussed here. In the following paragraphs, the most closely related computational approaches to knowledge generation and learning are described and compared with the algorithm presented in this paper.

CLARION The cognitive architecture CLARION (see, e.g., [16]) was designed to capture implicit and explicit learning processes in humans. The main assumption is that there are two different levels of learning: A subsymbolic level and a more explicit, declarative level. The subsymbolic, or 'bottom' level represents low-skill, often repetitive tasks where learning proceeds in a trial-and-error fashion. Knowledge on this level is typically not accessible, and it is difficult to express such skills with language. On the symbolic, or 'top' level, knowledge is directly accessible and can be expressed with language. This level typically represents more complex knowledge. It can be acquired by experience, but also by means of explicit teaching.

The input state is made up of a number of dimensions, and each dimension may specify a number of possible value or value ranges. Action selection takes place using RL in the bottom level, or by firing production rules on the top level. Which level is used is determined stochastically. After the action was performed, top and bottom levels are updated with the feedback received from the environment.

At the bottom level, the RL mechanism is implemented with a neural net. The input layer is constituted of the values of the input state. Three intermediate layers are used to compute Q-values (allowing memory of action sequences), while the fourth layer chooses an action according to standard reinforcement learning (similar to equation 2).

At the top level, the rule conditions are constructed out of the input dimensions, their consequents from actions available to the agent. The rules are, for compliance with the bottom level, implemented as network. Rule extraction,

specialisation and generalisation is determined by feedback from the bottom level: If there is no rule matching the current state and the action was successful according to some performance criterion, a new rule is created with the current state as the condition, and the performed bottom level action as consequent. If rules matching the current condition exist and the action was successful, the matching rules are replaced by a generalised version by adding another input element to the condition. The covered rules are deactivated, but might become reactivated if specialisation is applied to the new rule at a later stage. Conversely, specialisation means the removal of an input value from the condition and is triggered when the result of an action was not successful in the specified condition. Deactivated rules are reactivated if the specialised rule does not cover them any more. An information gain measure that estimates the performance of rules under different conditions serves as the success criterion.

Learning Classifier Systems Also Learning Classifier Systems (LCS) aim at the extraction of rules. The basic idea is to start with a set of initial rules (classifiers) and to evolve this set over time by application of mechanisms for modification, deletion and addition of new rules. Whereas earlier LCS, e.g. [5], relied mostly on the Genetic Algorithms paradigm newer versions have more in common with RL approaches and therefor also been described as generalised RL [12].

An LCS consists of a population of classifiers. A classifier contains a condition part, an action part, and an estimation of the expected reward. Typically, the condition part consists of the three basic tests 0 (property does not exist), 1 (property exists) and #. # represents a generalisation and stands for both 0 or 1. A classifier has one action as a consequent, but typically several classifiers match a condition in the environment and hence compete with each other. The action to be executed is the selected according to some RL mechanism (e.g. the ϵ-greedy policy).

Many LCS use a Genetic Algorithm to create new rules by selecting and recombine the fittest classifiers from the population (where fitness is, e.g., the expected reward received from the environment). A covering operator is called whenever the set of matching classifiers is empty. The operator adds a classifier matching the current situation with a randomly chosen action to the population. Sophisticated systems may limit the population size, and add corresponding eviction and generalisation procedures.

The newest family of classifier systems, anticipation-based classifier systems (ACS, e.g. [1]), does not rely on evolutionary methods. They extend the classifier representation with the description of the next state and build a model of transitions. A specialisation mechanism is applied when the classifier oscillates between correct and incorrect predictions, indicating that a splitting of the condition might improve the match. Generalisation is based on complex algorithms that estimate whether generalisation will result in an improvement (see also [12] for an overview of LCS).

Comparison RL is the most important aspect for generating action-centred knowledge in the related approaches as well as for the Bounded Rationality

Algorithm. Differences exist in the way such knowledge is used to build internal models of the environment:

- BRA does not start with a psychological model of skill acquisition as CLARION or no explicit model at all as machine learning, but a sociopsychological model of bounded rationality.
- The Bounded Rationality Algorithm uses a pure symbolic representation of conditions with simple first order predicate logic. CLARION has to transform them in a network structure, LCS in binary strings.
- CLARION modifies rules only after evaluation of bottom level actions; ACS compares prediction errors. BRA is much less sophisticated here, using a simple generate-and-test procedure to decide whether a rule should be specialised or generalised. If the test phase fails (possibly only after a long time when the environment changes), the generated rule is deleted again. CLARION as well as ACS keep detailed statistics and perform complex estimations to decide about generalisation and specialisation of specific rules.
- BRA starts with a state description covering all possible states and builds a model by searching heuristically through the space of possible state descriptions that can be expanded logically from the initial descriptor. In CLARION as well as ACS, it is not necessary to describe the state space fully. If new states are encountered, new rules are created on the fly. BRA is therefore much more sensitive to characteristics of the state space. For example, for state variables with large value spaces, specialised rules would be discovered only at later stages of the state expansion mechanism. Even if very fine-grained differentiations are useful, they might never be developed because descriptions generated on the path might not be immediately more successful than more general rules, so that the path is not further explored. However, BRA could be extended to cover initially only a small range of conditions, adding new attribute values dynamically as they appear in s.

7 Conclusion

In this paper, an algorithm that replicates the decision process of bounded rational actors has been described, formalised and demonstrated. It adds to systems based on reinforcement learning in the social sciences and combines elements of approaches as already used by CLARION and learning classifier systems. However, it is different from these approaches as it is less general than a cognitive architecture and explicitly built upon a sociopsychological approach to learning.

So far, the algorithm has been applied only to simple decision problems like that in the example. Future work will investigate whether the algorithms works as expected in more complex environments where a larger number of states and actions have to be learnt.

References

1. Butz, M.V.: An algorithmic description if ACS2. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.): Advances in learning classifier systems, Lecture Notes in Artificial Intelligence, vol. 2321, Springer, Berlin, 211-229 (2002)
2. Camerer, C., Ho, T.: Experienced-Weighted Attraction Learning in Normal Form Games, Econometrica, vol. 67, No. 4 (1999)
3. Erev, I., Roth, A. : Predicting how people play games: reinforcement learning in experimental games with unique mixed-strategy equilibria. American Economic Review 88, 848-881 (1998)
4. Grigerenzer, G., Goldstein, D. G.: Reasoning the fast and Frugal Way: Models of Bounded Rationality, Psychological Review, 103(4), 650-699 (1996)
5. Holland, J.H: Adaptation in natural and artificial systems: An introductory analysis with application to biology, control, artificial intelligence, University of Michigan Press, Ann Arbor (1975)
6. LeBaron, B, Arthur, W.B, Palmer, R.: Time series properties of an artificial stock market, Journal of Economic Dynamics & Control 23, 1487-1516 (1999)
7. Kahnemann, D., Tversy, A.: Prospect Theory: An Analysis of Decision under Risk, Econometrica 27 (1979)
8. Lopez, Lola L.: Psychology and Economics: Perspectives on Risk, Cooperation, and The Marketplace, Annual Review of Psychology, 45, 197-227 (1994)
9. Markham, A.B.: Knowledge Representation, Lawrence Erlbaum Associates, Mawah NJ (1999)
10. Roth, A., Erev, I.: Learning in Extensive Form Games: Experimental Data and Simple Dynamic Models in the Intermediate Run, Games and Economic Behaviour 6, 164-212 (1995)
11. Rubinstein, A.: Modeling Bounded Rationality, MIT Press, London (1998)
12. Sigaud, O., Wilson S.W.: Learning classifier systems: A survey, Soft Computing 11, 1065-1078, 2007
13. Simon, H.A.: A Behavioural Model of Rational Choice. In Simon, H.A.: Models of man, social and rational: mathematical essays on rational human behavior in a social setting, New York (1956)
14. Simon, H.A.: Rational Choice and the Structure of the Environment. In Simon, H.A.: Models of man, social and rational: mathematical essays on rational human behavior in a social setting, New York (1956)
15. Simon, H. A.: Bounded Rationality: Today and Tomorrow, Mind and Society 1(1), 25-39 (2000)
16. Sun, R., Slusarz, P.: The Interaction of the Explicit and the Implicit in Skill Learning: A Dual-Process Approach, Psychological Review, Vol. 112, No. 1, 159192 (2005)
17. Sun, R., Naveh, I.: Social institution, cognition, and survival: A cognitive-social simulation, Mind and Society, Vol.6, No.2, 15-142 (2007)
18. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, Cambridge, MA (1998)

Appendix: The complete algorithm

Notation

The greek letters $\gamma, \nu, \mu, \zeta, \chi$ are the parameters of the model
$C = \{C_1 \dots C_n\}$ are conditions that can be generated from the input dimensions S
A is the set of actions
R is the set defined by the mappings $r_i : C_i \rightarrow A$

Setup

Define the time discount for action updates γ
Define the update-cycle μ
Define the delete-cycle $\nu, \nu < \mu$
Define the cost of expansion ζ
Define the maximum number of states descriptions χ
Define the probability of switching the search path ρ
Define the search path $search\_path$
Define $expansions(r)$ as a function counting the number of expansions from r
Define $activations(r)$ as a function counting the times r matched a state
Define $parent(n)$ as the parent of a node n in the state-tree $T(R)$
Define $children(n)$ as a function returning all children of a node n in $T(R)$
Define $uniform(x \dots y)$ as a uniform random distribution in the interval $x \dots y$

Initialise $q(a) = 0, \forall a \in A$
Create the initial mapping $r_1 = \{C_1 \rightarrow A\}$ such that C_1 is inclusive of all states generated by the input dimensions S
Initialise $search\_path$ with $R = \{r_1\}$

Repeat
observe reward $p(t-1)$ received after executing a_{t-1}

update the average reward g at time t of all actions applied until t g_t:
$g_t = g_{t-1} + \frac{1}{2}p - g_{t-1}$

update the strength of action executed at $t-1$ using the action-value function:
$q(a_t) = q(a_{t-1}) + \gamma(p(t) - q(a_{t-1}))$

update the value of state-descriptor activated at time $t-1$:
$v(r,t) = v(r,t-1) + \frac{1}{2}(q(a_t) - v(r,t-1))$

update the activation-count:
$activations(r) = activations(r) + 1$

compute situation s

find the most specific state-descriptor $r_a \in search\_path$ matching s

determine selection probabilities of all actions $a \in A$ from r_a:
$pr_{a_{i,t+1}} = \frac{e^{q_{a_i} * \alpha}}{\sum_{j,j \neq i} e^{q_{a_j} * \alpha}}$

select action a_t from the resulting distribution and execute a_t

if $(\text{rest}(\frac{t}{\mu}) = 0$ and $|R| < \chi)$
determine which state-descriptor r_{expand} to expand:
$r_{expand} = \max h(r), \forall r \in search\_path$

if $(\zeta \times g_t \times expansions(r_{expand}) < v(r_{expand}))$
partition r_{expand} according to the expansion mechanism into
$R' = \{r'_0 \dots r'_n\}$; initialise the value of the new states with $v(r_{expand,t})$

append R' as children of r_{expand}

add R' to $search\_path$

update the number of expansions generated from r_{expand}:
$expansions(r_{expand}) = expansions(r_{expand}) + 1$
end if
end if

if $(\text{rest}(\frac{t}{\nu}) = 0)$
determine the most recent expanded node $r_{expanded}$ and its children:
$CH = \{ch_1, \dots ch_n\} \subset search\_path$ with $children(ch_i) = \emptyset$

$r_{expanded} = parent(ch_i)$

if $v(r_{expanded}, t) > \frac{1}{|CH|} \sum_{i=0}^{|CH|} v(ch_i)$ delete CH
end if

if $(uniform(0,1) > \rho)$
clear $search\_path$

find the most specialised state-descriptor with maximal value:
$r_{max} = \max v(r,t), \forall r \in R$, $children(r = \emptyset)$

add the path from r_1 to r_{max} to $search\_path$
end if
until end of simulation